Using Mobile Agents for Distributed Network Performance Management

Damianos Gavalas[1], Dominic Greenwood[2], Mohammed Ghanbari[1], Mike O'Mahony[1]

[1]Communication Networks Research Group,
Electronic Systems Engineering Department,
University of Essex, Colchester, CO4 3SQ, U.K.
E-mail: {dgaval, ghan, mikej}@essex.ac.uk
[2]Distributed Network Management and Agent Technology Research Group
Fujitsu Telecommunications Europe Ltd.,
Northgate House, St. Peters Street, CO1 1HH, Colchester, U.K.
E-mail: D.Greenwood@ftel.co.uk

Abstract. The intrinsic limitations of traditional centralised Network Management (NM), such as information bottlenecks and lack of flexibility, have encouraged a trend towards distributed management intelligence. Although several distributed NM architectures, exploiting the advantages of Mobile Agents (MA) have been recently proposed, when considering Network Performance Management (NPM) they fail to address scalability problems. In this paper, we describe a secure and fault-tolerant management framework based on MAs, which addresses these limitations by introducing two efficient, lightweight polling modes. Both real-time and off-line NM data acquisition is considered. An in-depth performance analysis of the introduced polling modes, in a data-intensive NPM application is also undertaken. The two modes are shown to outperform SNMP-based polling both in terms of response time and bandwidth consumption.

1. Introduction

The current state of the art in NM involves a management application (*manager*) and the managed entities (*agents*[1]), embedded within Network Elements (NEs). Management interactions make use of a centralised Client/Server (C/S) model, with the manager (client) collecting status data and setting control variables through the agents (servers). Communication between the managing and the managed parties is facilitated by NM protocols such as the Simple Network Management Protocol (SNMP) [1], part of the TCP/IP protocol suite and the Common Management Information Protocol (CMIP) [2] used in public telecommunication networks. Within these protocols, physical resources in a network are represented by managed objects.

[1] The term 'agent' here refers to static management agents, which should not be confused with Mobile Agents.

Collections of managed objects are grouped into tree-structured Management Information Bases (MIB)[2] following the Abstract Syntax Notation 1 (ASN.1) format.

Network Management Systems (NMS) based on this C/S archetype exhibit several drawbacks: due to rigid design time definitions, NMS functionality cannot be dynamically updated, whilst frequent polling of NEs for management data is known to result in substantial data transmission rates between the manager and agents. This creates a processing bottleneck at the manager host and adds a considerable strain on network throughput [6].

All these problems suggest distribution of management intelligence as a rational approach to overcome the limitations of centralised NM. The IETF has proposed an approach, known as RMON (Remote MONitoring) [4], which introduces a degree of decentralisation. In particular, RMON network monitoring devices (*probes*) collect management statistics from their local domain (e.g. an Ethernet segment), providing detailed information concerning traffic activity. However, this approach is expensive as it typically requires a stand-alone RMON compliant device (probe) in every network segment.

In terms of current research activities, Management by Delegation (MbD) [5] represents a first clear effort towards decentralisation. The idea of management distribution is taken further by solutions that exploit Mobile Agents (MA), which provide a powerful software interaction paradigm that allows code migration between hosts for remote execution. The data throughput problem can be addressed by delegation of authority from managers to MAs, which are able to filter and process data locally without the need for transmission to a central manager. This ability has attracted much attention to MA technology, with several Mobile Agent Frameworks (MAF) proposed for NM applications [7][8][9]. Neither [7] nor [8] though, consider remote processing issues and, as a result, the manager host still suffers from a computational burden induced by processing bottlenecks. These issues are addressed in [9].

It should be emphasised that all the aforementioned works involve frequent MA transfers, when the collection of management statistics is considered. In addition to the apparent network overhead, these frameworks are not scalable as they assume a 'flat' network architecture, i.e. a single MA is launched from the manager platform and sequentially visits all the managed NEs, regardless from the underlying topology. Thus, for large networks the round-trip delay for the MA will greatly increase, whilst the extracted statistics will not be accurate and reliable due to the non-negligible time intervals between the acquisition of each data sample for every NE. For such reasons, these MAFs are not appropriate for Network Performance Management (NPM), which represents the main focus of this paper.

NPM involves gathering and logging data generated by devices, which may be analysed off-line or in real-time. That process helps in measuring the performance, throughput and availability of network resources. The advantage of analysing the data in real-time is that it allows sophisticated NMSs to foresee a possible congestion or failure and take preventive measures before the actual error occurs. On the other hand, collected data may be used to build daily, weekly or monthly graphs/reports to assist the administrator in network planning. In such cases there is no need for real-time NM

[2] MIB is an SNMP term, which corresponds to the term Management Information Tree (MIT) used in CMIP. Hereafter, we ignore the difference for sake of simplicity.

data and, hence, an alternative and complimentary polling mechanism should be applied.

Therefore, we propose two MA-based polling modes intended to provide an efficient method for obtaining both real-time and off-line management data. In the first approach, called *Get 'n' Go* (GnG), used to collect real-time data, the network is partitioned into several domains and a single MA object is assigned to each of them. In every Polling Interval (PI), this MA sequentially visits all NEs within the network domain and obtains the requested information before returning to the manager. The second polling scheme, called *Go 'n' Stay* (GnS), targets the acquisition of data to be analysed off-line, where the need to obtain data in short time frames is no longer an imperative. Thus, we introduce a method where an MA object is broadcasted to all managed devices; the MA remains there for a number of PIs and collects an equal number of samples before returning to the manager. The infrastructure described in [9] has been extended to support the introduced polling modes.

The paper is organised as follows: Section 2 provides an overview of the MAF used to support this work, while Section 3 describes in detail the two introduced polling modes. Section 4 discusses how MAs can perform semantic compression of NM data, with a performance analysis given in Section 5 and experimental results reported in Section 6. Section 7 concludes the paper and suggests several topics for further work.

2. Mobile Agent Network Management Framework

The MA-based NM framework has been entirely developed in Java [12] as it offers the platform independence required for the management of distributed heterogeneous environments. This and other features, such the rich class hierarchy for communication in TCP/IP networks and its already wide acceptance for the development of distributed applications, positions Java as an ideal platform for MA-orientated management services. The key assumption of our approach is the presence of Java Virtual Machine (JVM) in every NE that is needed to host MAs as active processes. Current trends [13] indicate that Java-enabled network devices are beginning to emerge into the open marketplace.

Our framework consists of four major components [9], illustrated in Figure 1:

1. The Manager application;
2. The Mobile Agent Server (MAS);
3. The Mobile Agent Generator (MAG);
4. The Mobile Agents.

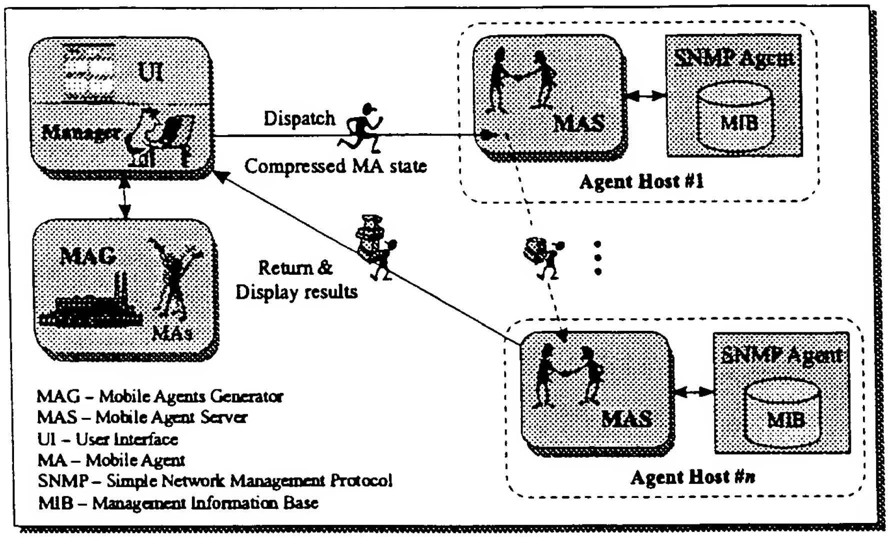


Fig. 1. The Mobile Agents-based Infrastructure

2.1. Manager Application

The manager application, equipped with a browser style User Interface (UI), co-ordinates monitoring and control policies relating to the NEs. Active agent processes are *discovered* by the manager, which maintains and dynamically updates a 'discovered list'.

2.2. Mobile Agent Server (MAS)

The interface between visiting MAs and legacy management systems is achieved through MAS modules installed on every managed device. The MAS resides logically above the standard SNMP agent, creating an efficient run-time environment for receiving, instantiating, executing, and dispatching MA objects, whilst protecting the system against external attack.

The MAS composes four primary components (see Figure 2):

- Mobile Agent Listener (MAL): a daemon that listens on well-known TCP and UDP ports for incoming MAs. Upon receiving an MA, its state is compressed and *de-serialised* [14].
- Security Component (SC): acts as the system's protective shield. The RSA algorithm [15] has been implemented to provide both ***authentication*** of incoming MAs and *encryption* of sensitive NM information.

- Service Facility Component (SFC): serves as an interface to the physical resources. The MA makes use of this component to interact indirectly with the SNMP agent and obtain system information. The acquired data are processed, if necessary, by an automatically invoked MA method.
- Migration Facility Component (MFC): upon an MA's request, it serialises and dispatches the MA object to the next host.

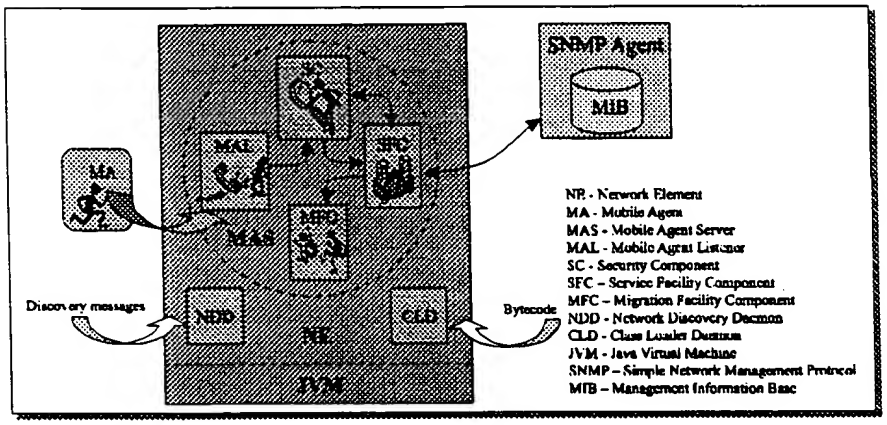

SNMP Agent
MIB
SC
MAL
SFC
MA
MAS
MFC
Discovery messages
NDD
NE
CLD
Bytecode
JVM
NE - Network Element
MA - Mobile Agent
MAS - Mobile Agent Server
MAL - Mobile Agent Listener
SC - Security Component
SFC – Service Facility Component
MFC – Migration Facility Component
NDD - Network Discovery Daemon
CLD - Class Loader Daemon
JVM - Java Virtual Machine
SNMP – Simple Network Management Protocol
MIB – Management Information Base

Fig. 2. The Mobile Agent Server structure

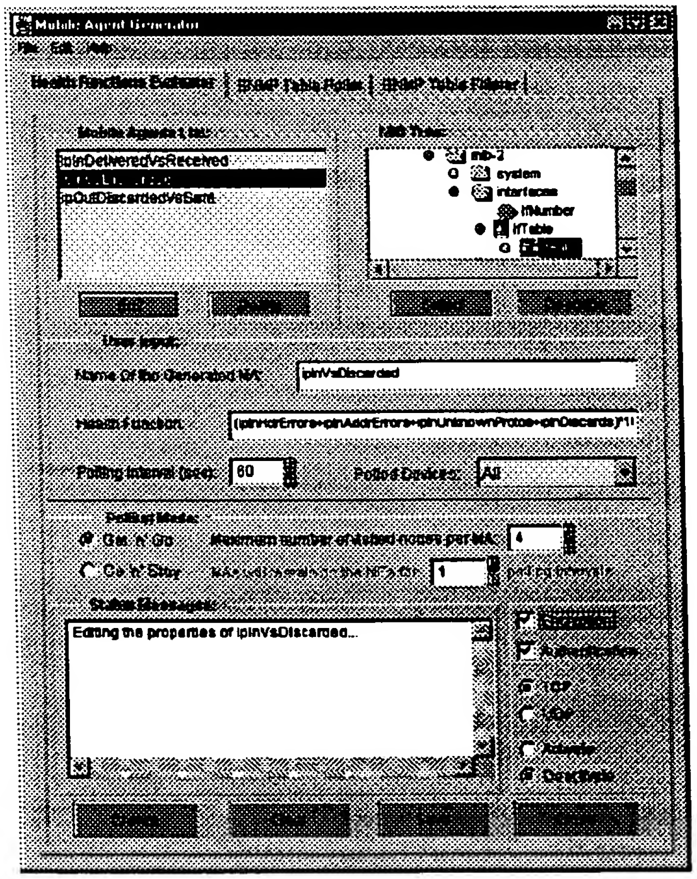
Two additional threads are present on each NE outside the boundary of the MAS: the Network Discovery Daemon (NDD) that allows the manager to discover active agent processes and the Class Loader Daemon (CLD), which receives and loads the MAs classes (bytecode).

Enhanced Security Pattern: The security pattern of [9] has been improved by introducing ***authorisation*** features that restrict the authority domain of visiting MAs on legacy systems. Specifically, the Java standard Security Manager (SM) has been extended to prevent MAs from directly reading/writing files, creating class loaders or sub-processes, exiting the MAS application, etc. This is achieved through registering the incoming MA threads to a given *thread group*, i.e. a batch of threads that eases the manipulation of active MAs. Based on the fact that an MA cannot change the thread group it belongs to, whenever a malicious action is detected, the SM checks the thread group of the action originating thread; when this thread belongs to the MAs thread group, the action is not permitted.

The only problem not addressed by the current implementation of the SM is that MAs cannot be restricted from entering an endless loop (consuming all CPU cycles), or creating thousands of new threads. However, in such an emergency situation, the SM could selectively destroy all threads not belonging to the MAS components' thread group. With these security extensions, we consider the network devices to be relatively safe from malicious MA attacks.

2.3. Mobile Agent Generator (MAG)

The MAG is essentially a factory for constructing customised MAs in response to service requirements. Such MAs may dynamically extend NMS functionality, post MAS initialisation, to accomplish management tasks tailored to the needs of a changing network environment.



ipInDeliveredVsReceived
ipOutDiscardedVsSent

Name Of the Generated MA: ipInVsDiscarded

Health Function: (ipInHdrErrors+ipInAddrErrors+ipInUnknownProtos+ipInDiscards)*1!

Polling interval (sec): 60

Polled Devices: All

Maximum number of visited nodes per MA: 4

1

Editing the properties of ipInVsDiscarded...

Fig. 3. Mobile Agent Generator User Interface

The MAG's operation is described in detail in [9]. However, its functionality has been extended so as to allow the operator (through the dedicated UI shown in Figure 3) to specify:

- Whether the constructed MA will be used for GnG or GnS polling;
- The polling frequency (i.e. the PI's duration);
- The transmission protocol to be used for the MA transfers (either TCP or UDP);
- Whether MAs authentication and data encryption are applied or not.

It should be emphasised that the transfer of the MA bytecode is performed only *once*, through broadcasting it to the active MASs at MA construction time. Thereafter, the transfer of persistent state, obtained from serialising the MA instance, is sufficient for the MAS entities to recognise the incoming MA and recover its state. This approach relieves the need to transfer MA bytecode to the same NEs during every PI. In contrast, [7] and [8] apply a policy that requires the transfer of both the MA's bytecode and persistent state, resulting in a much higher demand on network resources, as code size is typically much larger than state size [11].

Pre-defined MA property sequences are stored in configuration files and parsed at manager initialisation to instantiate corresponding MA objects. These properties may be modified subsequent to agent instantiation; the modifications will take instant effect.

2.4. Mobile Agent

From our perspective, MAs are Java objects with a unique ID (consisting of the manager host network address and an individual sequence number), capable of migrating between hosts where they are executed as separate threads and perform specific management tasks.

The MA model described in [9] has been refined by introducing an MA *superclass*, which provides some root attributes: an itinerary, a data folder where management information is stored, a problem folder used to report faults. In addition, we have implemented several service-oriented classes that extend the MA superclass. These in turn, are sub-classed by MA classes created by the MAG. Superclass methods are either invoked at MA instantiation, arrival, departure or used to control interaction with polled devices. This flexible hierarchical approach minimises MA bytecode and eases the creation of service specialised MAs.

Also to protect the MAs against tampering, sensitive MA properties (e.g. ID, itinerary, etc.), may be specified only once, when the MA is instantiated. If a malicious host attempts to modify this information, an exception is thrown.

Itinerary, data and problem folders have been implemented as Java *vectors*, i.e. dynamic arrays. At each visited NE, a data sample is appended to the data folder whereas the local NE address is removed from the itinerary vector. In this way and by applying data aggregation methods, as described in Section 4, we prevent the MAs state from growing too rapidly.

The MA's state information is compressed (using the Java *gzip* utility) before being transferred to the next destination host to minimise communication overhead.

2.5. Tolerating Node Failures

An MAF should be able to survive failure scenarios by improving the infrastructure's fault tolerance in terms of securing MA migrations.

Scenario 1: An MA should adapt to unexpected situations, such as the failure of a host MAL thread. In this case, if TCP is used for MA transfers, the TCP connection establishment fails (Figure 4(a)). The MA's *onFailMigration*() method is then automatically invoked to record the unreachable host's name into the MA's problem folder and retrieve the next destination host from the itinerary vector (Figure 4(b)). The MA will then migrate to this host (Figure 4(c)). When returning to the manager host, the MA reports the failed device threads to the manager application, which in turn will take any necessary regenerative actions.

If UDP is the transport protocol choice, the detection of a failed MAL thread is more complex as the UDP datagram carrying the MA's state would simply be lost. A feasible solution to this problem would be to enforce the destination host to issue an acknowledgement when successfully receiving an incoming MA. In case of a fault, the acknowledgement would never reach the originating node and after a given time interval the MA image would be transmitted to its next destination. However, such an approach would create an additional traffic load, thereby reducing the benefit of UDP's lightweight nature.

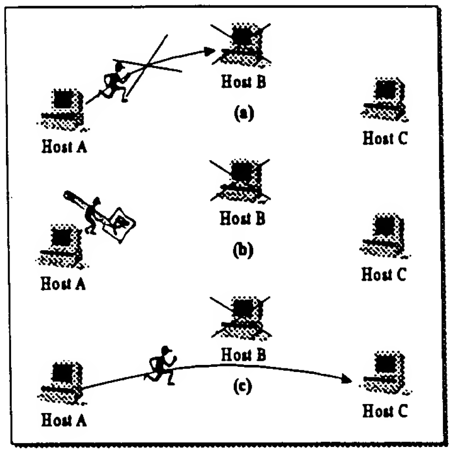


Fig. 4. MA reaction to the detection of a failed MAL thread

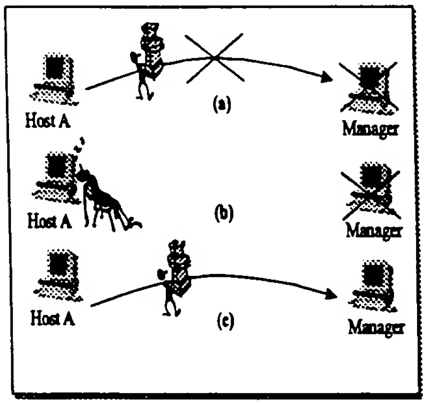


Fig. 5. The specific case of MAL thread failure on the manager host

Scenario 2: A second scenario would be a fault in the MAL thread of the manager host itself (Figure 5(a)), which represents a special case. Since loss of management data carried by an MA should be avoided, an alternative approach to scenario 1 is employed. Upon detecting a fault, the MA will 'sleep' for a given interval (Figure 5(b)) and then resume execution in order to retry the connection. If the manager application recovers before a pre-determined number of retries elapses, the MA is transferred (Figure 5(c)), otherwise it is disposed of locally.

3. Polling Modes

3.1. Get 'n' Go Polling Mode

Traditional SNMP-based polling, which is intrinsically centralised, involves a flood of request/response messages as shown in Figure 6(a). This naturally leads to a significant proportion of available bandwidth being used for management data.

On the other hand, recently reported NM MAFs [7][8][[9] assume a flat network structure. Hence as the number of managed devices grows, the network becomes increasingly unmanageable. This is a consequence of having a single MA responsible for obtaining NM data from each device in every PI (see Figure 6(b)), causing serious scalability problems. In addition, the order in which managed nodes are visited is arbitrary. This represents a significant problem when the management of remote LANs is considered, as a travelling MA may have to be transferred several times across expensive and low-bandwidth WAN links during its lifetime.

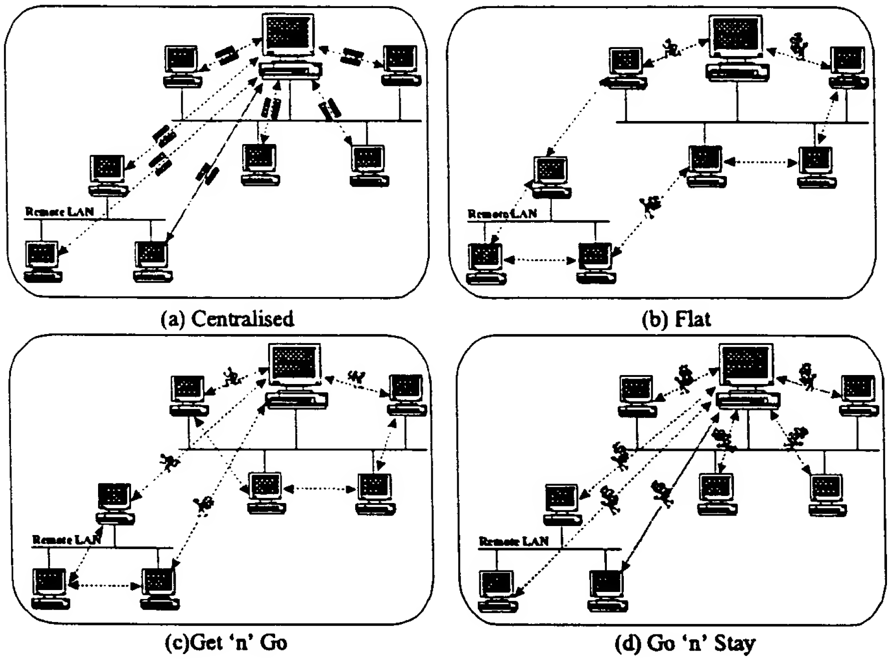


Fig. 6. Alternative approaches to polling.

As previously mentioned, the concept behind GnG polling is to partition the managed network into several logical/physical domains. The partitioning criteria are specified by the administrator and may correspond to: (i) the number of nodes assigned to each MA, (ii) the physical distribution of polled devices, or (iii) a hybrid of these two approaches. The number of MAs required per PI is automatically evaluated and their individual itineraries either manually or automatically specified. For instance, in Figure 6(c), an MA object polls the devices of the remote LAN, whereas a second MA is assigned to the network segment local to the manager host. With the GnG approach, the launched MA objects travel and perform their management tasks in parallel, whilst the number of devices they visit is limited to help minimise the overall response time. This factor suggests GnG as a suitable polling scheme for the acquisition of real-time data.

As the number of devices assigned per MA is reduced, the volume of MAs required to conduct polling is increased and the journey time of each of them decreased. Nevertheless, the manager needs more time and consumes more CPU cycles to instantiate, launch and receive back this larger number of MAs, whilst the communication overhead imposed by polling increases. Thus, an optimal solution has to be found as discussed in Section 6.

In terms of implementation, GnG polling is carried out through *Polling Threads* (PT). PTs are started and controlled by the manager application; each of them corresponds to a single polling instance. When started, PTs retrieve polling definitions and schedules from their associated configuration files and poll NEs on a regular basis. Specifically, PTs instantiate and launch the required number of MAs (supplied with their corresponding itinerary) and then sleep for one PI. When this period elapses the same process is repeated. Meanwhile, a manager's listener daemon receives the MAs that return to the manager carrying their collected data.

PTs may be synchronised such that they are not initiated simultaneously. This ensures that the traffic around the manager host will be distributed over time. In addition, the discovery of a new active agent process triggers an automatic re-evaluation of the required number of MAs per PI and their itineraries.

A graphical table (within the manager's UI) displays and allows modifications to existing PT properties, such as PT activation/de-activation, polling frequency and the number of devices assigned to each MA.

3.2. Go 'n' Stay Polling Mode

GnS polling introduces an alternative approach to NPM, targeting data intended for off-line analysis. This reduces the number of MA transfers whilst the volume of data carried by each MA increases. Hence, the proportion of *useful* management information within the MAs state is substantially increased, compared to the other approaches where a large number of MA transfers may be required to obtain a few data samples.

Specifically, every PT broadcasts at regular intervals an MA object to all agent hosts. The MAs then remain active on the hosts for p PIs (where p is specified by the administrator). At the end of each PI they obtain a sample of the requested data set and encapsulate it into their state. The MAs then sleep for one PI and awaken to obtain another sample. When the p PIs elapse, the MAs return to the manager to deliver the acquired samples (see Figure 6(d)). Meanwhile, PTs suspend execution for a duration given by the product of PI and the number p of PIs that MAs remain on the managed devices ($PI \times p$). When this period expires, they resume operation and the process is repeated.

Clearly, there is a trade-off between bandwidth consumption and response time. As p increases, so does the response time to the manager. However, as the MA transfers become sparser the network overhead imposed by polling reduces. If, for instance, p=100, MA objects will be broadcasted every 100 PIs. When changing to p=50, the MA transfers are doubled but the response time is halved. In the extreme case that p=1, the response time is minimised and GnS mode becomes similar to SNMP-based polling and identical to GnG, when each MA is assigned to a single device.

The administrator is given the option to modify the value of p and also dynamically change the polling mode from GnS to GnG and vice versa, depending on the managed network traffic conditions and the types of management information required. When changing from GnS to GnG, automatic network domain segregation takes place.

4. Semantic Compression

Polling is a frequent operation in NM as there are often several object values that require constant monitoring. Cases often occur however, where one or two Management Information Base (MIB) variables are not a representative indicator of system state and hence an aggregation of multiple variables is required (known as a *health function*) [16]. For instance, *five* MIB-II objects [3] are combined to define the percentage *E(t)* of IP output datagrams discarded over the total number of datagrams sent during a specific time interval,

$$E(t) = \frac{(ipOutDiscards + ipOutNoRoutes + ipFragFails) * 100}{ipOutRequests + ipForwDatagrams} \quad (1)$$

where MIB-II is an example of a MIB being supported by all the SNMP managed NEs.

In SNMP, the manager can only retrieve atomic MIB object values, hence all the operating agents will receive *five* requests corresponding to the *five* object values appearing in Eqn. (1). The value of *E(t)* is then computed by the manager when all object values have been returned in separate response packets.

On the other hand, the MAs constructed by the MAG tool are able to perform *semantic compression* of management information. Thus, the value of *E(t)* is computed locally, with a single value returned to the manager station. Hence, the manager is relieved from processing NM data, while the MAs state size remains as small as possible.

A major contribution made by this MA-based approach to NM is the *domain level* view of the managed network that mobility intrinsically implies. This is due to the multi-node movement that MAs often undertake, which allows for a superjacent level of data filtering. For instance, the manager requests the minimum value of *E(t)* within the network: In such a case, each MA would compare the value of *E(t)* acquired from the current host with the minimum value recorded so far and retain the lesser of the two.

Aside from the use of health functions, many other techniques could be applied to achieve data compression. For example, MAs could be customised to apply intelligent search methods to the extraction of data from large SNMP tables.

5. Performance Analysis

As evidenced in [11], a key factor affecting performance is the overhead induced by transport layer protocols. In particular, TCP is more reliable and yet traffic intensive resulting from its connection-oriented nature. In contrast, connectionless UDP sacrifices reliability in favour of a lightweight communication mechanism. In our implementation, the choice of the transport protocol is left to the network operator.

First, let us consider SNMP-based polling for the computation of Eqn. (1). If $S_{req/res}$ is the average request/response size (in application layer), and polling of n devices for v operational variables is applied, the wasted bandwidth for i PIs would be:

$$B_{SNMP} = 2.(S_{req/res} + O_T).n.v.I \tag{2}$$

where O_T is the overhead imposed by the transport protocol (UDP in the SNMP case).

When GnG or GnS polling is employed, a considerable compression in data volume is achieved at the source, as explained in the preceding section. Thus, assuming an MA with compressed bytecode of size C and compressed state information of size S_h (when migrating from the hth host), the resulted overhead for GnG polling when segmenting the network into d domains would be,

$$B_{GnG} = n \bullet (C + O_T) + d \bullet i \bullet \sum_{h=1}^{h_{tot}} (S_h + O_T), \text{ where } d \le n \text{ and } h_{tot} = \left\lceil \frac{n}{d} \right\rceil + 1 \tag{3}$$

as the v variables are aggregated into one. The first term of the equation describes the overhead imposed when broadcasting the MA code to all MASs, whilst the second represents the bandwidth consumed by the MA state transfers between the manager and the polled devices (each MA is assigned to $\left\lceil \frac{n}{d} \right\rceil$ NEs, hence h_{tot} transitions in total, including the return to the manager). Thus, for large v and i, GnG mode is less bandwidth intensive than SNMP-based polling. However, there is of course a threshold value of the number v of requested operational variables, below which the efficiency of the MA approach is lost in favour of SNMP-based polling [10].

The MA's state size, S_h, is given by,

$$S_h = S_{per} + S_{d_h} + S_{it_h} \tag{4}$$

where S_{per} is the static/permanent information, while S_{d_h} and S_{it_h} are the data and itinerary vector's size respectively, when leaving the hth host. As explained in Section 2.4, S_{d_h} increases and S_{it_h} decreases for larger values of h. By defining the *constant* difference, dS, in MA's size after visiting each NE: $dS = (S_{d_h} + S_{it_h}) - (S_{d_{h-1}} + S_{it_{h-1}})$, Eqn. (4) becomes,

$$S_h = S_{per} + S_{d_1} + S_{it_1} + h \bullet dS\,, \quad \text{or} \quad S_h = S_1 + h \bullet dS \tag{5}$$

and Eqn. (3) changes to:

$$B_{GnG} = n \cdot (C + O_T) + d \cdot i \cdot \left[(S_1 + O_T) \cdot h_{tot} + \sum_{h=1}^{h_{tot}} h \cdot dS \right],\ d \le n, \qquad (6)$$
$$h_{tot} = \left\lceil \frac{n}{d} \right\rceil + 1$$

Similarly to GnG, the network overhead for GnS polling would be,

$$B_{GnS} = n \cdot (C + O_T) + n \cdot [2 \cdot (S_1 + O_T) + dS'] \cdot \left\lceil \frac{i}{p} \right\rceil \qquad (7)$$

as MAs remain on the managed devices for p PIs collecting an equal number of samples resulting in a state size increment of dS'. For a large p, the number of MA transfers is minimised and GnS mode becomes the most lightweight polling approach in terms of bandwidth consumption.

As far as the transport protocol overhead O_T is concerned, when TCP protocol is in use, a single MA transfer requires a TCP connection to be set-up and subsequently released, i.e. 6 TCP messages, including acknowledgements. Each TCP message is prepended by a 20 byte TCP header and encapsulated into an IP datagram that introduces a 20 byte header[3], namely: O_T = 240 bytes. For UDP, a single packet is used to transfer the MA's state. The UDP header is 8 bytes long and encapsulated into an IP datagram, hence: O_T = 28 bytes.

6. Experimental Results

The experimental testbed comprises a network of several Solaris and WinNT machines. We assume the physical distribution of managed devices in the network as arbitrary to these experiments.

6.1. Response Time

In Figure 7, GnG polling mode is compared to an SNMP implementation [17] in terms of the response time for the acquisition of the health function result of Eqn. (1). Figure 7(a) shows that the flat approach (using one MA) does not scale well as the number of NEs increases. This makes it necessary to partition the managed network into several domains in order to maintain lower response times over the SNMP-based polling scheme.

[3] The measurements have been performed over an Ethernet LAN, which introduces an additional 26 bytes overhead on each frame. However, the overall size is slightly larger as there is a minimum frame size requirement of 72 bytes.

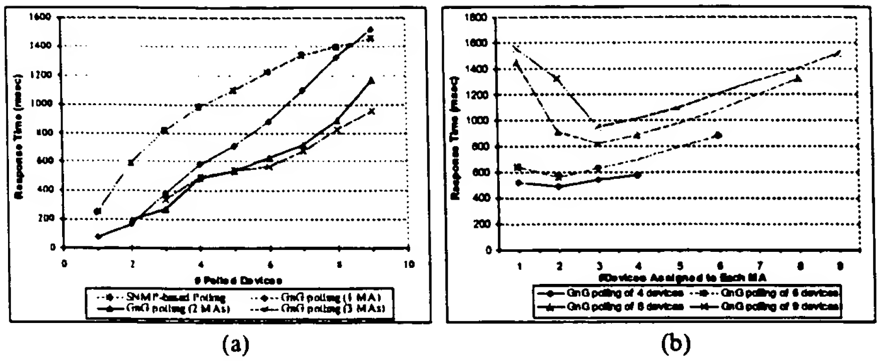
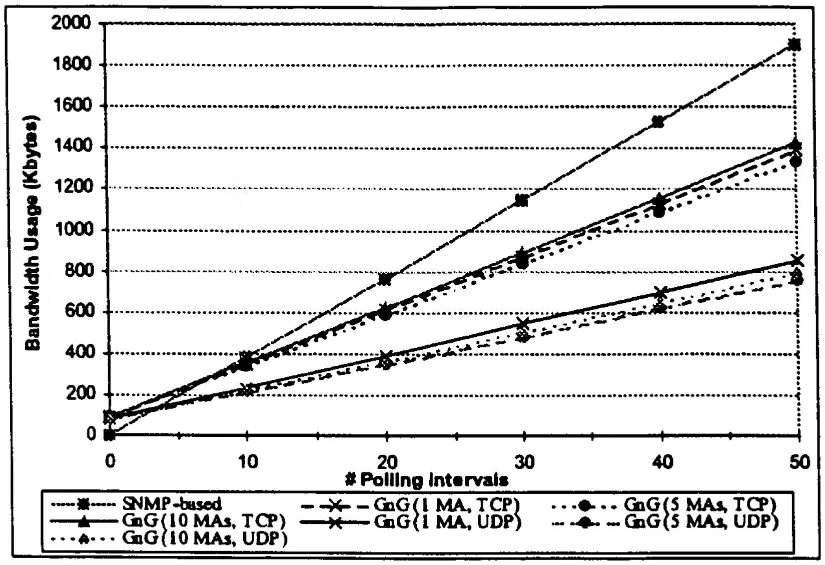


Fig. 7. Response time of (a) SNMP-based vs. GnG polling, (b) GnG polling as a function of the number of devices assigned to each MA object.

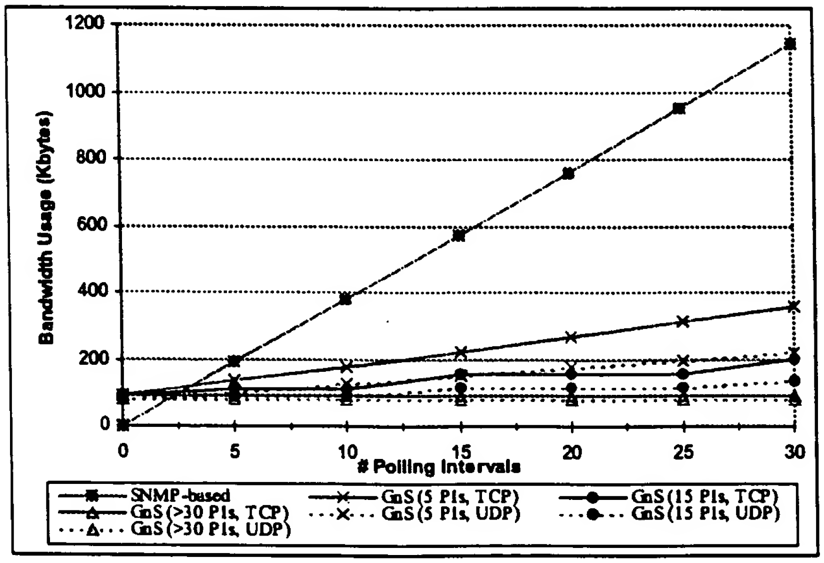
Depending on the managed network size, the optimum number of domains can be determined from the minimum point of the corresponding curve of Figure 7(b). From the discussion in Section 3.1, when assigning a small number of devices to each MA, i.e. when launching a large number of MAs per PI, the overall response time increases. Specifically, although the individual journey times decrease, the time required for the manager to instantiate, launch and receive back these MAs dominates. Thus, for a network of six devices, the response time is minimised when each MA is assigned to two NEs, i.e. the network is segmented into three domains. Hence, the manager application (through the PTs) should autonomously adapt the number of domains according to the current number of managed devices. This issue is currently under investigation.

6.2. Bandwidth Consumption

Figures 8(a) and 8(b) illustrate a traffic overhead comparison between SNMP-based and GnG or GnS polling schemes respectively, according to equations (2), (6) and (7). For GnG and GnS polling, results for both TCP and UDP are shown.



(a)



(b)

Fig. 8. Bandwidth consumption for SNMP-based against (a) GnG and (b) GnS polling modes.

For this experiment, $S_{req/res}$=50 bytes, C=1.6Kb, S_i=205 bytes, while n=50 devices and v=5 operational values. Also, the differences in state sizes were measured to be $dS = 3$ bytes and $dS' \approx 5$ bytes/sample. Hence, the initiation point for GnG/GnS polling depends on the transport protocol used and is derived from the first term of Eqn. (6) and (7). Clearly, both GnG and GnS modes represent a notable improvement over SNMP-based polling. However, the transport protocol choice has a very serious effect on bandwidth usage, with UDP significantly outperforming TCP.

An interesting result from the GnG polling experiment is that using a single MA does not necessarily prove less traffic intensive than using a larger number of MAs (see Figure 8(a)), although in the former case the overall number of MA transitions is minimised. This observation is explained by the summation: $\sum_{h=1}^{h_{tot}} h \bullet dS$, included in Eqn. (6). In the flat approach (d=1), the MA object visits n hosts, with its state rapidly growing, causing scalability problems (for a large n). In contrast, when partitioning the network into several domains, each of the MAs return to the manager to deliver the results before its state becomes too large. Again, there is an optimum number of MAs that minimises the bandwidth consumption, dependent on the network size.

Figure 8(b) confirms that GnS mode becomes more attractive as the number PIs (p) that MAs remain on devices is increased. Hence, the choice of the transport protocol is crucial only for a small p, i.e. when MA transfers are relatively frequent. As the value of p increases the associated curves for TCP and UDP are shown to converge.

It can therefore be concluded that the selection of the appropriate polling mode (and the associated parameters, including the transport protocol), is a compromise between network overhead, response time and reliability, depending primarily on the type of management data to be collected.

7. Conclusions

A framework that exploits the capabilities of MAs in NPM and improves on the scalability of recently reported MAFs, has been presented. Security and fault-tolerance features have been integrated into the framework, and the efficiency requirement has been answered by introducing two novel polling modes.

First, we introduced the GnG mode that may be used for obtaining real-time NM data as overall response time is minimised. This method may also be preferable when considering the management of remote LANs. Concerning the off-line analysis of management data, we propose the GnS mode. In this approach, MAs collect a larger amount of data before returning to the manager leading to a direct reduction in the number of MA transfers.

In both cases, techniques for semantic compression of NM data are applied to reduce the bandwidth usage. The performance analysis and results indicate a significant improvement in both response time and traffic overhead when comparing the introduced polling modes to traditional centralised polling. The choice of transport protocol used for MA transfers has proven a critical factor regarding the polling modes' performance.

Current work addresses:

- Extensions to MAG functionality, such as constructing MAs able to filter SNMP tables;
- Optimisation of MAs itinerary to minimise GnG polling response time;

- Minimisation of MA state.

References

[1] Case J., Fedor M., Schoffstall M., Davin J., "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

[2] ISO/IEC 9596, Information Technology, Open Systems Interconnection, Common Management Information Protocol (CMIP) – Part 1: Specification, Geneva, Switzerland, 1991.

[3] McCloghrie K., Rose M., "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1213, March 1991.

[4] S. Waldbusser, "Remote Network Monitoring Management Information Base", RFC 1757, February 1995.

[5] Yemini Y., Goldszmidt G., Yemini S., "Network Management by Delegation", Proceedings of the 2nd International Symposium on Integrated Network Management (ISINM'91), 1991.

[6] Baldi M., Gai S., Picco G.P., "Exploiting Code Mobility in Decentralised and Flexible Network Management", Proceedings of the 1st International Workshop on Mobile Agents (MA'97), pp. 13-26, 1997.

[7] Ku H., Luderer G., Subbiah B., "An Intelligent Mobile Agent Framework for Distributed Network Management", Proceedings of the IEEE Global Telecommunications Conference (Globecom '97), pp. 160-164, 1997.

[8] Susilo G., Bieszczad A., Pagurek B., "Infrastructure for Advanced Network Management based on Mobile Code", Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'98), pp. 322-333, 1998.

[9] Gavalas D., Greenwood D., Ghanbari M., O'Mahony M., "An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology", accepted to IEEE International Conference on Communications (ICC'99), 1999.

[10] Gavalas D., Greenwood D., Ghanbari M., O'Mahony M., "A Hybrid Centralised - Distributed Network Management Architecture", accepted to the 4th IEEE Symposium on Computers and Communications (ISCC'99), 1999.

[11] Fuggetta A., Picco G.P., Vigna G., "Understanding Code Mobility", IEEE Transactions on Software Engineering, 1998, vol. 24, no. 5, pp. 342-361.

[12] Sun Microsystems: "Java Language Overview – White Paper" [On-line] (1998), URL: http://www.javasoft.com/docs/white/index.html.

[13] Sun Microsystems, Jini Technology White Papers (1999), URL: http://sun.com/jini/whitepapers/.

[14] Sun Microsystems, Java Object Serialisation Specification, Feb. 1997.

[15] Rivest R.L., Shamir A., Adleman L., "A Method for obtaining Digital Signatures and Public-Key Cryptosystems", Communication of the ACM, 21(2), Feb. 1978.

[16] Goldszmidt G., "On Distributed Systems Management", Proceedings of the 3rd IBM/CAS Conference, 1993, URL: http://www.cs.columbia.edu/~german/papers.html.

[17] AdventNet, URL: http://www.adventnet.com/.

dblp.uni-trier.de

Damianos Gavalas

List of publications from the DBLP Bibliography Server - FAQ

Coauthor Index - Ask others: ACM DL - ACM Guide - CiteSeer - CSB - Google

		2002
4	EE	Damianos Gavalas, Dominic P. A. Greenwood, Mohammed Ghanbari, Mike O'Mahony: Hierarchical network management: a scalable and dynamic mobile agent-based approach. Computer Networks 38(6): 693-711 (2002)
		2000
3		Damianos Gavalas, Dominic P. A. Greenwood, Mohammed Ghanbari, Mike O'Mahony: Deploying a Hierarchical Management Framework Using Mobile Agent Technology. IS&N 2000: 333-348
		1999
2	EE	Damianos Gavalas, Dominic P. A. Greenwood, Mohammed Ghanbari, Mike O'Mahony: Using Mobile Agents for Distributed Network Performance Management. IATA 1999: 96-112
1		Dominic P. A. Greenwood, Damianos Gavalas: Using Active Processes as the Basis for an Integrated Distribution Network Management Architecture. IWAN 1999: 199-211

Coauthor Index

1	Mohammed Ghanbari	[2] [3] [4]
2	Dominic P. A. Greenwood	[1] [2] [3] [4]
3	Mike O'Mahony	[2] [3] [4]

DBLP: [Home | Search: Author, Title | Conferences | Journals]
Michael Ley (ley@uni-trier.de) Wed Oct 13 20:14:34 2004